BISOFT-28
Estructuras de Datos 2

Diseño de Algoritmos II

Programación Dinámica & *Backtracking*

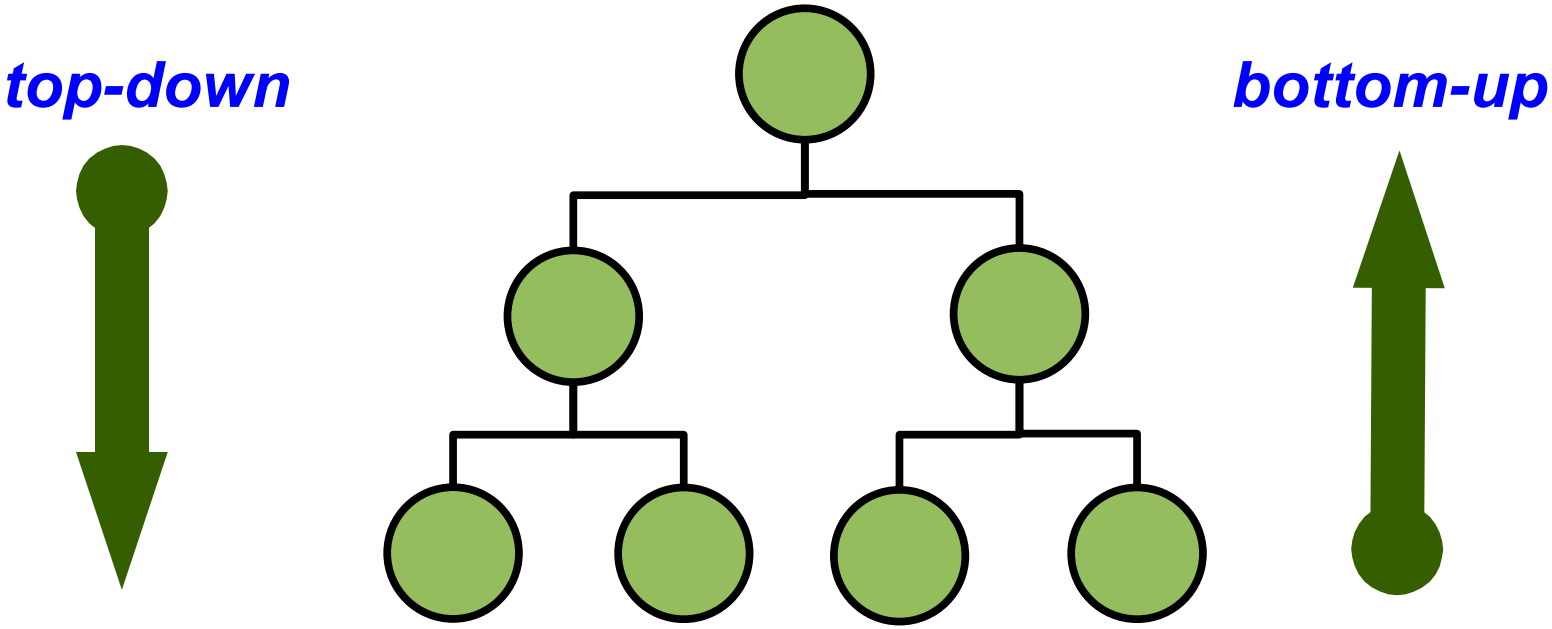
Instructor Esteban Meneses

Programación Dinámica

- Programación Dinámica **vs** Divide y Vencerás.
- **Divide y vencerás**:
  - Eficiente.
  - Sub-casos independientes.
  - Si estos sub-casos se traslapan, no es eficiente.
- **Programación Dinámica**:
  - Evita cálculos **duplicados**.
  - Usa **tabla** de casos conocidos.
  - La tabla se utiliza para ir llenando **casos** cada vez mayores.

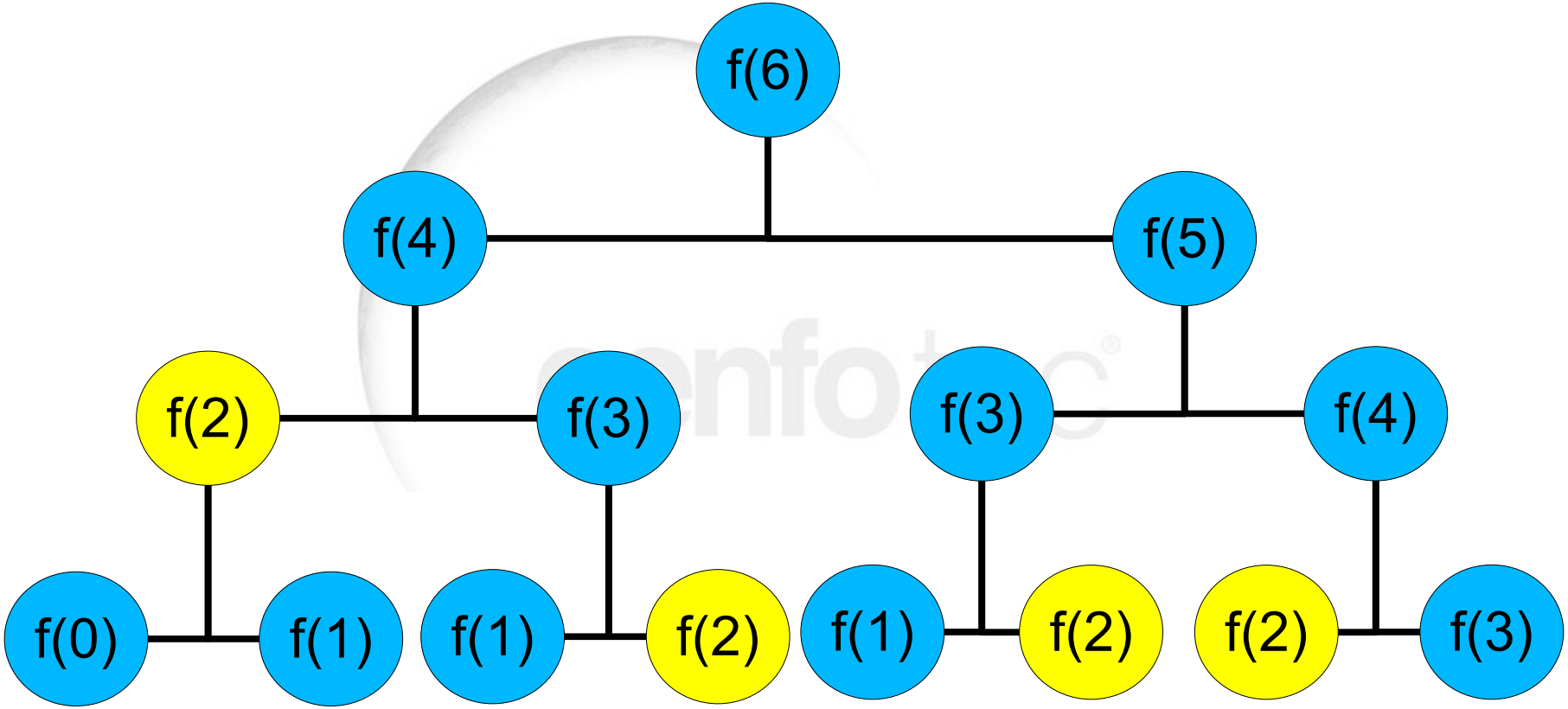
Programación Dinámica

- Otra forma de ver la **diferencia**:
  - Divide y vencerás: refinamiento progresivo, ***top-down***.
  - Programación dinámica: técnica ascendente, ***bottom-up***.



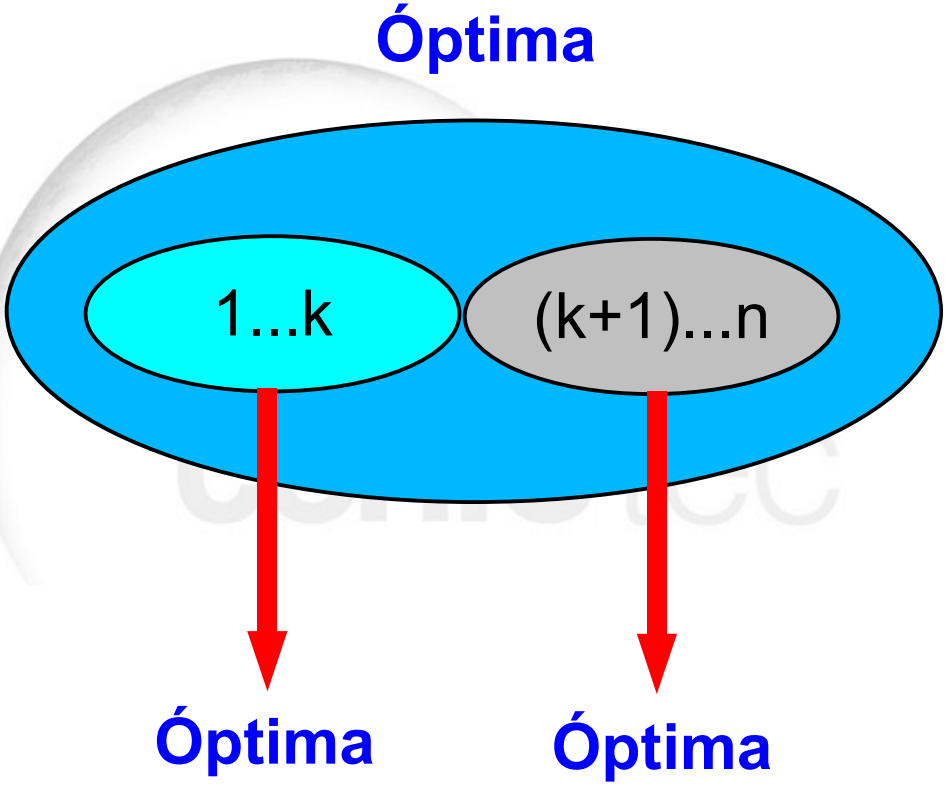
Programación Dinámica

- Ejemplo de **Recursión**.
  - Cálculo de los números de Fibonacci.



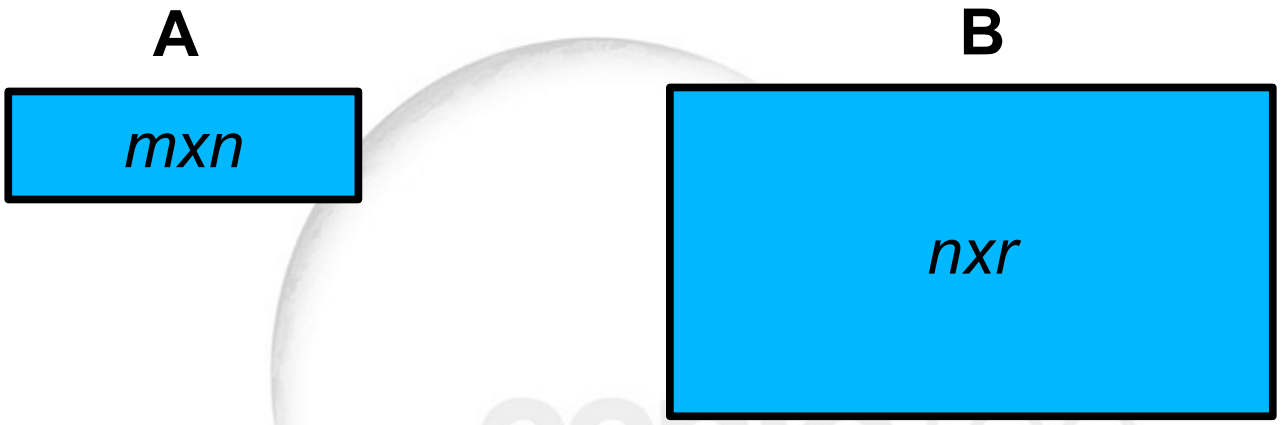
Principio de Optimalidad

- En una sucesión óptima de decisiones, toda sub-secuencia debe ser también óptima.

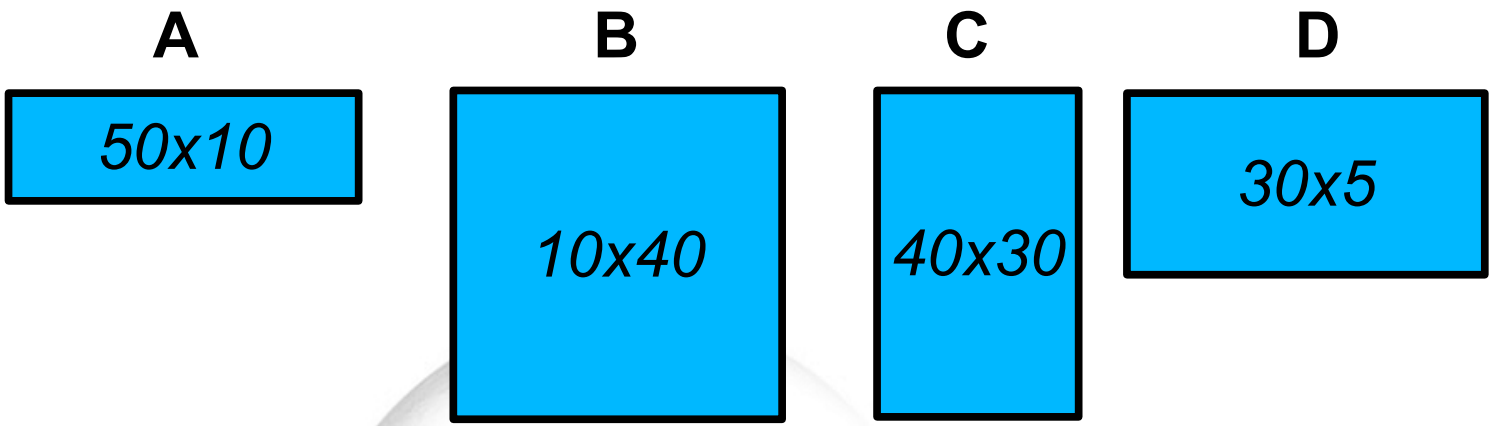


Caso 1: Multiplicación Encadenada de Matrices

- A: *mxn*, B:*nxr*, la multiplicación de AxB toma *mnr* **operaciones**.



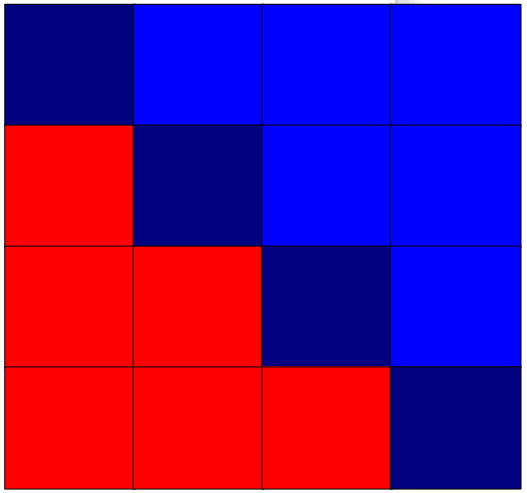
Caso 1: Multiplicación Encadenada de Matrices



- Posibles órdenes:
  - A((BC)D) : **16 000** operaciones.
  - A(B(CD)) : **10 500** operaciones.
  - (AB)(CD) : **36 000** operaciones.
  - ((AB)C)D : **87 500** operaciones.
  - (A(BC))D : **34 500** operaciones.

Caso 1: Multiplicación Encadenada de Matrices

- Matriz m: contiene en la posición (i,j) la cantidad de **operaciones** necesarias para multiplicar las matrices desde la *i* hasta la *j*.
- En cada casilla se guarda *k*, el valor de **corte** óptimo.
- El vector *d* guarda las **longitudes** de las matrices.



- $m_{i,i} = 0$
- $m_{i,i+1} = d[i-1]*d[i]*d[i+1]$
- $m_{i,j} = \min_{i<=k<j}\{m_{i,k} + d[i-1]*d[k]*d[j] + m_{k+1,j}\}$

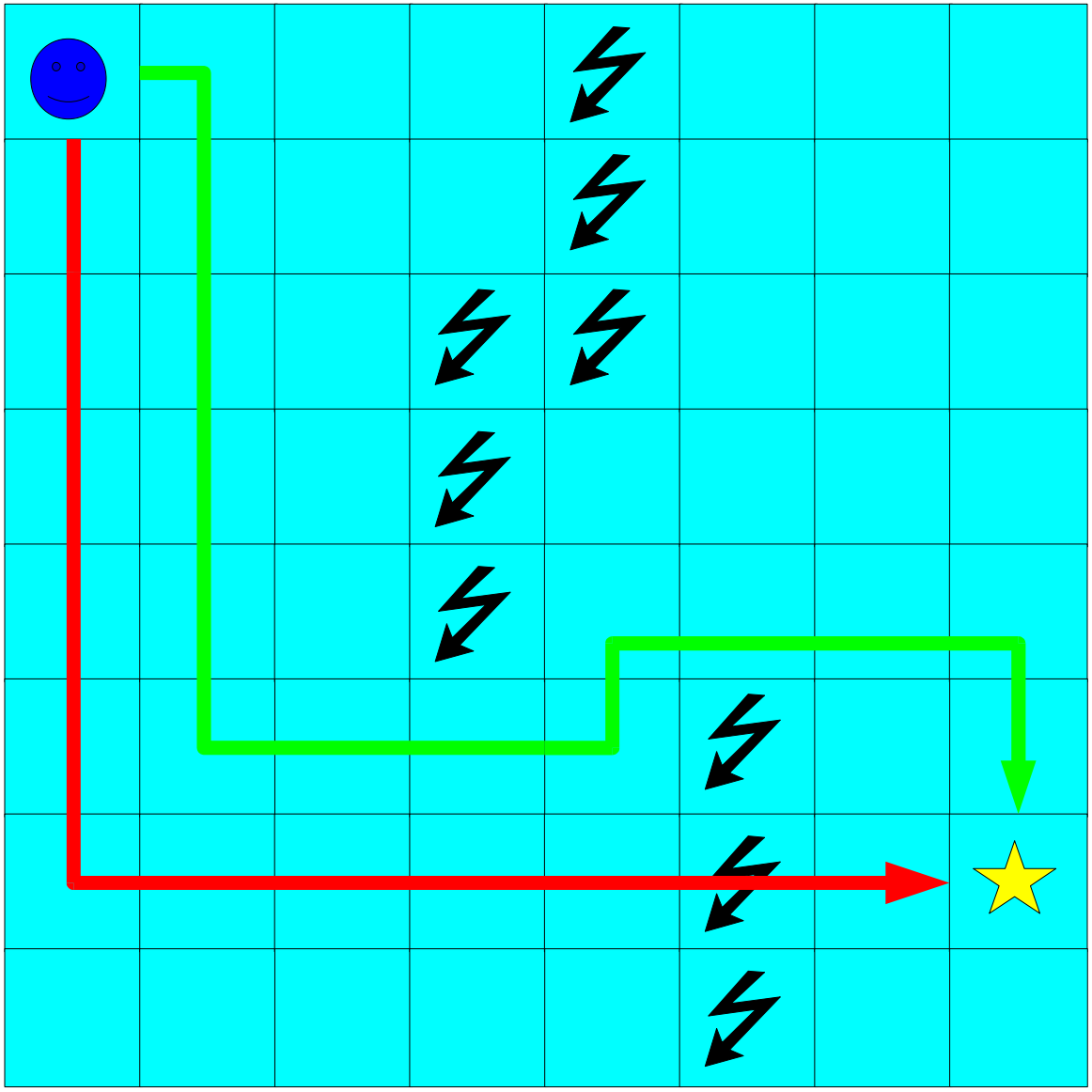
Backtracking

- Voracidad **vs** *Backtracking.*
- *Backtracking* es una técnica que **reconsidera** decisiones tomadas anteriormente.
- Pueden existir **retrocesos** en la búsqueda de la solución final.

Caso 2: Navegación de robot con A*

- Encontrar la ruta **más corta**.
- Utiliza un **heurístico**.
- Se elige el nodo que tenga el **mínimo**:
  $f(n)=g(n)+h(n)$
- $g(n)$ es el **costo** de llegar al nodo n.
- $h(n)$ es el **costo heurístico** del nodo n al final.
- Heurístico utilizado: distancia **Manhattan** (número de cuadros para llegar hasta el objetivo)

Caso 2: Navegación de robot con A*



Caso 3: El Problema de Reconstrucción

- Dados *n* puntos en el eje x, se calculan todas las **distancias** entre los puntos.
- El problema de **reconstrucción** consiste en encontrar los puntos, dadas las distancias.

FIN